

Modeling cloud resources using Machine Learning

Josep Ll. Berral, Ricard Gavaldà, Jordi Torres

Universitat Politècnica de Catalunya and Barcelona Supercomputing Center

Jordi Girona 31, 08034 Barcelona, Spain

{berral,torres}@ac.upc.edu, gavaldà@lsi.upc.edu

Abstract

Cloud computing is a new Internet infrastructure paradigm where management optimization has become a challenge to be solved, as all current management systems are human-driven or ad-hoc automatic systems that must be tuned manually by experts. Management of cloud resources require accurate information about all the elements involved (host machines, resources, offered services, and clients), and some of this information can only be obtained a posteriori. Here we present the cloud and part of its architecture as a new scenario where data mining and machine learning can be applied to discover information and improve its management thanks to modeling and prediction. As a novel case of study we show in this work the modeling of basic cloud resources using machine learning, predicting resource requirements from context information like amount of load and clients, and also predicting the quality of service from resource planning, in order to feed cloud schedulers. Further, this work is an important part of our ongoing research program, where accurate models and predictors are essential to optimize cloud management autonomic systems.

Keywords: 68T05 - Learning and Adaptive Systems, 68M14 - Distributed Systems, 68M20 - Performance Evaluation, 93A30 - Mathematical Modeling

1 Introduction

Cloud Computing, the new paradigm of distributed and clustered computing, has become a crucial model for the Internet architecture towards the externalization of information and IT resources for people and enterprises. It brings the possibility of offering “everything as a service” (platform, infrastructure and services), allowing companies to move their IT, previously in private owned data-centers, to external hosting. But the control and optimization of these infrastructures is not easy, as many elements and actors are involved on its development, performance and power and resources consumption.

Driving the cloud must take into account

very different elements, with different behaviors and interactions. The main goal of the manager is to maximize the revenue: As cloud customers, we want to run our services on the cloud, obtaining good performances toward our clients using our web services; as cloud resources providers, we want to run as many services provided by our customers in our resources, obtaining revenue for each run service; also, as resource owners, we want to reduce the running costs of resources, by not providing more resources than the minimum required. In order to make decisions organizing services and resources we dispose of low-level data (resource, power and operating system monitors) and high-level data (user behavior and service performance like uptime, response time and availability). While other works try to model client behaviors (high-level user data), we model the high-level versus low-level data in order to find the relations between the resource usage and assignment, the service performance and client satisfaction, and power consumption.

Here we present the cloud scenario (e.g. grid based data-centers) as a set of resources and a set of web services. Each resource (CPU, memory, bandwidth or disk) has a maximum quota of usage and its energy requirements. Also each service has resource requirements, a load (clients using the service) per time unit, performance measures determined by the agreement between provider and customer, and an execution reward. The management and decision making consists on placing each service on a hosting machine that assures the availability of the required resources, and that the other services being hosted in the host do not compete excessively for these available resources. A good strategy to be applied is “consolidation”, attempting to set the maximum number of services in the least viable amount of hosting machines, so the number of running machines and resources is minimized.

The challenge is how to do that executing the maximum number of services without compromising performance and user satisfac-

tion (quality of service). A way of solving this challenge is obtaining as most information from the system status as possible, and predicting as much information from each possible action to perform, being able to choose the best action based on predicting its consequences. As many of the parameters and functions involved in this optimization problem are unknown a priori, data mining and machine learning methods are ideal for obtaining the required information. Using the ability to create models from past experiences we use basic known data to create a model for each element in the system (an application type, a workload, a physical machine, a high-level service requirement). The system can then use these models to make the appropriate choices towards optimizing the cloud management, and keep them updated from new test and real runs.

As a case of study and introduction of the cloud scenario, we show here the modeling of the different main resources found in hosting machines, and their relations with the typical quality of service (QoS) measures. For a given web service application, models can predict features such as minimum CPU usage, minimum memory occupation and bandwidth status, and dependence on workload volume. For quality of service elements, such as response time, a model can predict their dependence on the resources allocated to the task and the low-level monitored quantities. All in all, the problem to solve is to decide the effects of resource allocations on hosts towards services QoS.

This work is organized as follows: Section 2 presents previous work in this area. Section 3 describes the architecture of the scenario. Section 4 shows the models and experiments for cloud components. Section 5 describes the models and experiments for quality of service prediction. Section 6 explains the uses of the models obtained. Finally, Section 7 summarizes conclusions and future work.

2 Related Work

Currently there are several work modeling the cloud in ad-hoc manners, using knowledge from experts. Works like Chase et al. [6] presented MUSE, a framework for modeling and autonomically control cloud hosting centers and its policies. Their approach is based on an economical managing for scheduling jobs to resources, where hosts and jobs bid for resources, and elasticity is allowed taking into account penalties for not fully attended resources. A similar way to model the cloud is presented in Goiri et al.

[8], where each policy to be applied on jobs and resources is represented by a set of conditions with rewards and penalties, based on the job objectives and the resource capabilities. Their modeling is focused on the Service Level Agreement (SLA) between the resources provider and the job customer. Also, in our previous work [3] we modeled a data-center with a mathematical program, including some resource behaviors and jobs requirements as constraints to be satisfied.

Some previous approaches used machine learning techniques on data-center management, focusing on policy selection. Approaches like Tan et al. [12], Tesauro et al. [13] and Kamitsos et al. [10], use Reinforcement Learning algorithms to manage resource allocation and power consumption, specifically techniques like Q-learning, SARSA and Markovian Decision Processes, better explained in Sutton et al. [11], all in order to select policies to be applied at each time. Works like Dhiman et al. [7] applies ML to control resources and power, selecting policies for specific resources like hard disk and network states. Also, Alonso et al. [2] applies ML and regression methods to learn to detect memory bugs from hosting machines, and predict the time to crash of it.

In other works like our previous ones [5, 3] we applied ML models to predict CPU behaviors and allocate services to host machines, minimizing used machines. In fact this current work is an important part of the complex mathematical program introduced in [3], where jobs in a data-center are scheduled optimizing revenue, power and SLA, with a set of constraints defined by the system itself and by data-center owners and users. This mathematical program requires accurate models of all the important elements of the system in order to find realistic optimal policies and solutions. So here we deepen on resource modeling in order to learn all the dimensions of the system and improve these decision makers.

2.1 Motivation All the current approaches using ML for resource management, as far as we know, are oriented towards learning the characteristics of specific components or towards learning the consequences of using policies on determined states of the cloud or its elements. Here we focus on the learning of resources model as a way to understand the consequences of reaching determined states and applying determined allocation policies. The final goal is to supply accurate on-line information to decision makers. Also data mining and machine learning are used

to model web service users. We avoid this kind of knowledge at this time focusing on a more low level knowledge. Obviously, in a more advanced stage of this work, information about users and web service structure can provide useful details for enhancing these behavior models, but at this moment we want to explore the basis yet to be determined.

3 Background: The Cloud Scenario

In this work we focus on a set of techniques for modeling Cloud resources, having enough information to manage and schedule jobs on host machines in the most proper way. From each job (web-services in this case) we want to discover how much resources it will demand to the cloud, given the characteristics of its clients. Also we want to discover how giving more or less resources affects each job taking into account its load and machine hosting it, so we can decide how much resources we give to each one and where we allocate it. How services work on the cloud is explained in the following subsection.

3.1 Architecture of Services Consider a typical commercial “cloud”, where customers have their data, applications and web-services running on it, without the need of knowing details of the infrastructure supporting them. Customers pay providers (owners or managers of the infrastructure) to keep their stuff on the cloud, and providers must ensure that this data and services have enough resources to be held and run properly. Clients are those users that will access to this data and services, paying for it to the customers (owners of data and services). Clients must be served quickly, without interruption, without error messages, etc., in order to be satisfied and trust (and pay) the customer for it; so the provider must give the data and web-services the required amount of resources to reach a good quality of service (QoS), so he is paid by the customer according to a service level agreement (SLA) defining this QoS.

Figure 1 shows the structure of the cloud. The provider has a set of data-centers, each one containing physical computing machines. Each physical machine contains resources and holds data and services from different customers. Thanks to virtualization technology, each customer is provided a virtual machine (VM), giving him the sensation of having a physical machine (PM) for his own; also data and services are isolated, bringing security and independence to services of different customers. Finally, inside each virtual machine, customers have their data

and services deployed.

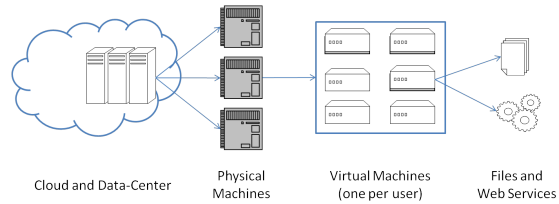


Figure 1: Commercial hosting infrastructure

The task of the provider is to let a VM to each customer, adjust the VM to the required resources for the data and services given the agreed QoS, and put each VM into a PM capable to provide these resources. Further the provider goal is to obtain the maximum profit from each VM execution and reduce to the minimum the cost of running resources (energetic, maintenance, etc.). For this, “consolidation” is a usual technique consisting on filling all the VMs in the minimum PMs, shutting down the empty ones, reducing the energetic consumption. So if the provider manages to adjust the required resources of each VM to the minimum instead of over-reserve resources, he can consolidate even more without degrading in excess the QoS.

To adjust the resources we must know about how granting or withdrawing them to VMs affect to the QoS. Here we depict the basic resources found in a data-center: CPU, memory and Input/Output devices (including storage devices). Web services, the kind of services used in this work, are applications on the web, usually working upon a system stack formed by the Operating System, the web server program, the web application program and the data base service. This stack is what the VM contains, and customers add their services and data on the web application program and data bases. A typical stack is e.g. GNU/linux OS + apache server + PHP + MySQL. Each stack and each customer service will require different amount of resources to serve with the desired QoS (see Figure 2).

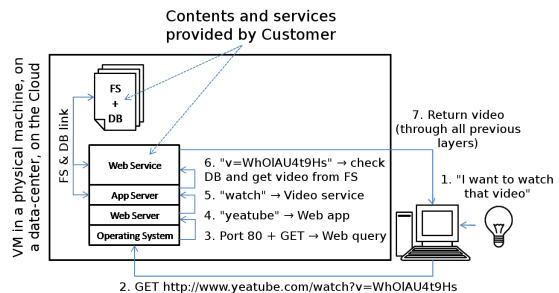


Figure 2: Commercial hosting infrastructure

Typical measure of Quality of Service on web sites are the average response time for web queries, the web site uptime, the ratio of satisfactorily solved queries, etc. Here we focus on the response time (RT), a measure affected directly by the amount of resources given to the VM and the amount of requests and clients received by the web-service. If a service receives less resources than the required for a given load, it will return a slower reply to the user web query; but giving a service more resources than needed will not imply it to reply quicker.

3.2 Information retrieval on DCs We want to model the usage of CPU, memory and I/O (network usage) against the load received towards a typical web service. For this, we can monitor load and resources from real hosting machines and real load, obtaining the following set of information/attributes: timestamps; number of requests and clients; average response times, bytes, CPU time usage and bandwidth per request; and usage of the CPU, memory and network resources. All this attributes are summarized on Table 1.

This information can be grouped in two sets of data: Load $\langle requests, clients, bytespr, timepr \rangle$ and Resources $\langle \{cpu, mem, net\}pm, \{cpu, mem\}vm, r\{cpu, mem\}vm \rangle$; also a response variable $RTpr$. The input in our problem is the characterization of the *Load* to be received. The modeled variables are the demanded resources, with the final goal of modeling the response variable *average Response Time per request*, using the information about load and resources. Further, other useful information or attributes can be derived from the obtained from the system monitors, for the prediction purposes. Some examples are:

- $\Delta T_{t-1,t}$: Time elapsed from last time unit (previous measures and current measures).
- $(\Delta)Load_{t-1}$: Any load or difference of load respect the last time unit.
- $(\Delta)Resources_{t-1}$: Any resource or difference of resources respect the last time unit.

3.3 Experimentation Sources The experiments to test this approach have been performed obtaining data from real workloads applied to real hosting machines, and then using this information in a simulated a data-center in order to check the scalability of the method. All the following experiments and data about behaviors and response times, have been obtained

from a full recreation of the LiBCN'10 workload [4] on the data-center provided by RDLab-UPC [1], with Intel Xeon 4 Core machines running at 3Ghz and with 16Gb RAM, running jobs of kind [Apache v2 + PHP + MySQL v5] in a virtualized environment [Ubuntu Linux 10.10 Server Edition + VirtualBox v3.1.8]. The workload have been properly scaled on some experiments according to the workload instructions in order to recreate different load volumes.

4 Resource Modeling

4.1 CPU Prediction The first resource to model is the CPU usage, as the main bottle-neck in a host related to load volume, so we try to learn the amount of demanded CPU by a VM given the characteristics of the incoming requests. Each request must be attended by the web service inside the VM, so depending on the amount of requests the VM will demand more or less CPU to process them. The number of requests, the average "time per request" obtained from the usual behavior and the expected average size of the incoming requests are the basic data we dispose for this CPU demand.

The expected number of requests $E[requests]$ can be obtained by observing the number of requests of the last time units, or knowing a priori the usual amount of requests given the week day and time hour, among other techniques; the expected bytes per request $E[bytespr]$ can be obtained or modeled by having the averages of bytes per request; and the CPU time per request $E[timepr]$ can be obtained also by measuring the average CPU time spent per request on a VM running alone in a test run without interferences of other VMs or jobs. All the other variables have been dismissed as they will not be available at the time of using this model, they have been considered independent to this specific problem, or after some tests they showed very low relevance.

By modeling the amount of CPU given the load in void (without other VMs or jobs in the Physical Machine) we obtain the natural CPU demand function of the VM and its contained web-services. This model will also include the CPU overhead provided by the virtualization platform. This predicted information will be necessary to learn the same behavior under stress, where VMs will compete for the same CPU resources of the PM (see next sections). At this time, the function to be learned is:

$$f_{cpuvvm}^{learned}(E[requests], E[bytespr], E[timepr]) \rightarrow E[cpuvvm]$$

<i>time</i>	Time-stamp for current measure.
<i>requests</i>	Number of requests for the current time unit.
<i>clients</i>	Number of clients (unique IP requests) for the current time unit.
<i>RTpr</i>	Average response time per request for the current time unit.
<i>bytespr</i>	Average bytes per request for the current time unit. Depends on each element requested.
<i>timepr</i>	Average CPU time per request for the current time unit. Depends on each kind of request done, run alone with no stress, as the minimum CPU required by the request.
<i>mbpspr</i>	Average network speed (mbps) per request for the current time unit.
<i>cpuv</i>	Average CPU demanded by the VM for the current time unit.
<i>memv</i>	Average memory demanded by the VM for the current time unit.
<i>cpup</i>	Average CPU occupied in the PM for the current time unit.
<i>memp</i>	Average memory allocated in the PM for the current time unit.
<i>rcpuv</i>	Average CPU given to the VM in the PM for the current time unit.
<i>rmemv</i>	Average memory given to the VM in the PM for the current time unit.
<i>nKBin</i>	Total KBytes read from network in the current time unit.
<i>nPktIn</i>	Total packets received from network in the current time unit.
<i>nKBOut</i>	Total KBytes sent to network in the current time unit.
<i>nPktOut</i>	Total packets transmitted to network in the current time unit.

Table 1: Attributes obtained from monitoring a host machine

In order to learn this function, we used a M5P algorithm [9]. The relation between requests, bytes per requests and time per requests seemed to be non-linear, as two of their parameters are averages laying over the third. But the algorithm M5P, a decision tree holding linear regressions on its leaves, is able to approximate this non-linear relation by parts. M5P details:

- Maximum instances per leaf: $M \in [50, 70]$
- Training Dataset: 3968 instances
- Model selection process: Random split
(Training 66%, Test 34%)
- Validation Dataset: 7528 Instances
- Error for $CPUVM$: $MRE = 0.164$; $MAE = 2.35\%$ CPU; $\sigma = 4.511$ (Dataset range $[2.37, 100.0]$ % CPU)
- Number of regressions obtained: 7

Figure 3 shows the model selection test and the validation test. The tree of regressions obtained shows that *requests* and *timepr* are the most relevant variables by driving the tree. It explains how requests and the CPU time per requests increase the amount of CPU required by the VM, something we expected and very reasonable.

4.2 Memory Modeling The second resource to model is the memory used by the VM. In most web servers and application servers the usage of memory is greedy and demand only grows until reaching a limit, and then memory

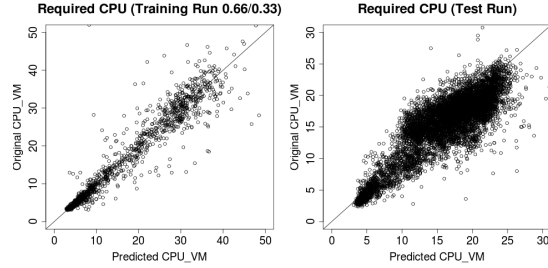


Figure 3: Prediction of VM CPU demand

space is reorganized by flushing caches or freeing unused elements. So the amount of memory demanded by the VM depends basically on load and time. Figure 4 shows the memory filling and reorganization on a apache web server given different amounts of memory limit.

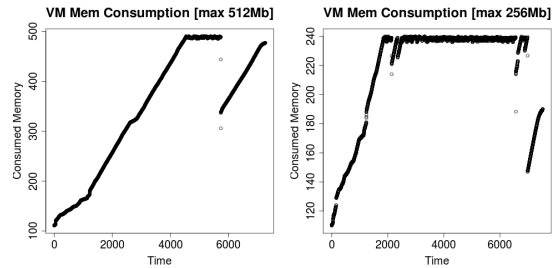


Figure 4: Typical VM Memory Behavior

Knowing how web servers usually work, each new client (web user) makes the web server and application server to allocate memory for his transactions, and also caches grow with each new request until reaching the memory limit

when memory allocated for old clients and unused cached elements is freed. In order to predict memory we should know how much memory we had previously allocated for the VM, and the expected amount of load incoming. So it can be learned as a function of the elements representing load ($E[requests]$, $E[bytespr]$, $E[timepr]$, $E[mbps]$, ...) and the previous memory status ($memvm_{t-1}$) and the time t of last memory measure. Being $memvm_{t-1}$ the amount of memory used at the previous observation, and ΔT the amount of time elapsed since then, the function to learn at this moment is:

$$f_{mem}^{learned}(E[requests], E[bytespr], E[timepr], E[mbps], memvm_{t-1}, \Delta T) \rightarrow E[memvm]$$

First experiments trying to find a function that fits the sample data, using Linear regression methods like LinReg, M5P or even SVMs, showed models incompatible between different time intervals ΔT , as each model fitted the average of ΔT , and new measures with extreme time intervals missed the target, and M5P tended to create a branch for each seen ΔT . After some observations and conclusions, we discovered that the relation of all our input data is not linear, as $memvm_{t-1}$ and ΔT followed a polynomial relation, as the weight of $memvm_{t-1}$ could vary depending on the age of the measure ΔT .

In order to confirm this suspicion, adding a new attribute $memvm_{t-1} \cdot \Delta T$ on the datasets, linear regression method fits adequately independently of the ΔT factor and correcting previous experiments without this attribute. Figure 5 shows the result of applying a Linear Regression as a memory learner, also details of the training and testing are shown as follows:

- Training Dataset: 7808 instances
(2408 instances [$\Delta T = 10$ sec], 4032 [$\Delta T = 10$ min], 1376 [$\Delta T = 1$ hour])
- Model selection process: Random split
(Training 66%, Test 34%)
- Validation Dataset: 243 Instances
(All of them $\Delta T = 5min$)
- Error for $memvm$: MRE = 0.0127; MAE = 4.3963 MBytes; $\sigma = 8.3404$ (Dataset range [124.2, 488.4] MBytes)

An interesting thing is that the function obtained in the regression shows that the expected VM Memory is almost always the previous memory value plus an increase depending on the load:

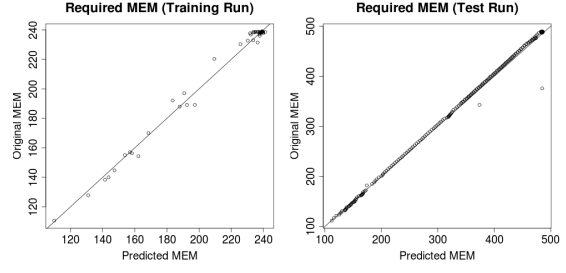


Figure 5: Prediction of MEM VM demand (TR: $\Delta T \in [10s, 10min, 1h]$; TS: $\Delta T = 5min$)

$$E[memvm_t] = memvm_{t-1} + f(Load_t, \Delta T, memvm_{t-1} \cdot \Delta T)$$

Although that, in this learning process we had to add external knowledge in order to obtain a good model (“memory occupation depends on time”), and the interesting thing would be to let the learning algorithm to discover it by itself.

4.3 Bandwidth Prediction The third resource to model is the network used by the physical machine, measures either by the amount of bytes or the number of packets entering and exiting the machine. A typical configuration in virtualization technologies is the shared or bridged network interface, so all traffic from VMs is directed to the physical network interface using NAT or a “virtual name” is added to the physical interface for each VM, so it has different Hardware Addresses one for each VM. This implies that all VMs dump and receive data directly from the same physical interface (PM traffic becomes the sum of all VMs network packets + PM network control packets).

Another typical configuration, in this case for commercial data-centers, is to place disks, file systems (FS) and data bases (DB) outside the computing hosts 6. This means that data and disk images for VMs are outside the PM and placed in hosts dedicated to serve data only. This implies that the usage of disk requires usage of network. Also, the data-center manager emits continuously control packets and synchronization packages in order to keep control of hosts, and keep VM disk usage possible and information updated.

So an important detail when collecting data from network is to detect when a control synchronization packets arrive messing statistics and causing that two examples of the same load with different network usages. For this time a solution found is to collect and aggregate data in a big enough time span to include at least

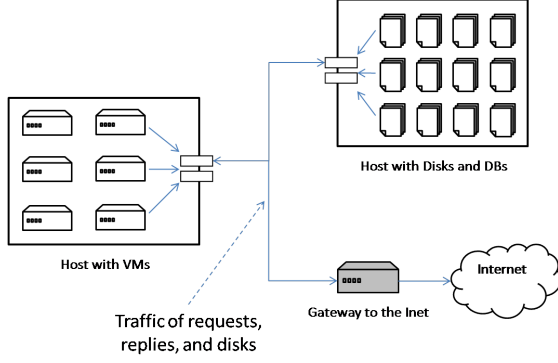


Figure 6: Dedicated host with disks and DBs

a control/sync flank with all the normal traffic (with the current system using SunGrid + NFS, instead of using measures each 10 seconds we take measures each 30 seconds).

So knowing the volume of data the host will generate over the network is also an important issue to take into account. Low load imply the network is usable and VMs can deliver the responses to requests at the moment, and high loads can saturate networks and requests or responses can not be delivered to the VMs. From the information about expected load, a function predicting the number of packets the VM and/or PM will generate in and out from can be learned, and from this the number of upstream and downstream Kbps and other derived measures. The function to be learned, using the appropriate time aggregation of data, is:

$$f_{bwd}^{learned}(E[requests], E[bytespr], E[timepr], E[mbps]) \rightarrow \langle E[\#PktIn], E[\#PktOut] \rangle$$

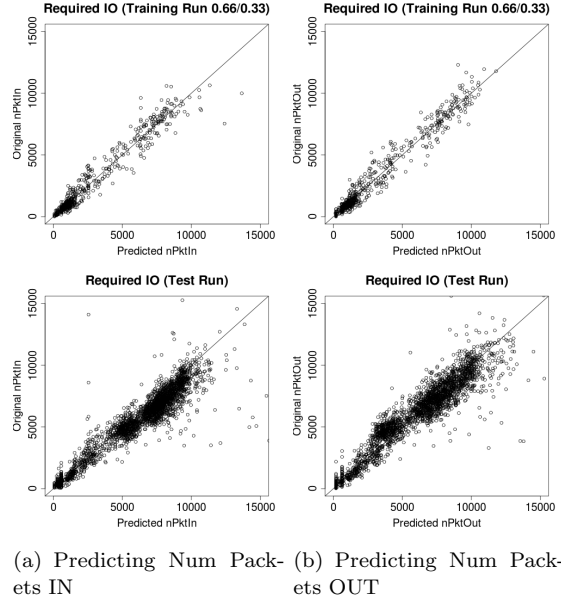
In this situation, the expected speed velocity $E[mbps]$ can be measured from the current time period or expected from known properties of the network.

In order to learn this function, we selected again a M5P algorithm among other tested methods (like LinReg or SVMs). We take the advantage of relation between the number of requests and the other parameters again to find a function that fits or approximates their relation. The details of the M5P usage are shown as follows:

- Maximum instances per leaf: PktIn, PktOut $M = 30$
- Training Dataset: 1623 instances
- Model selection process: Random split (Training 66%, Test 34%)

- Validation Dataset: 2423 Instances
- Error for #PktIn: MRE = 0.193; MAE = 926 Pkts; $\sigma = 1726$ (Dataset range [56,31190] #Packets)
- Error for #PktOut: MRE = 0.184; MAE = 893 Pkts; $\sigma = 1807$ (Dataset range [25,41410] #Packets)

Figure 7 shows the model selection test and the validation test.



(a) Predicting Num Pack- (b) Predicting Num Pack-
ets IN ets OUT

Figure 7: Prediction of PM Bandwidth demand

5 Interaction Modeling

Traditionally data-center planners and schedulers consider a maximum known amount of resources for each VM, and VMs are allocated in PMs with often underloaded resources, so the interference and competition for resources is minimal. Systems like operating systems, web-servers, and other “request based” components have different behavior being or not under stress. They usually behave in a regular way until the demand for resources is higher than the available. In data-centers and VMs stress creates a competition for resources, increasing overheads due to CPU context change, renewing caches, memory pages swapping, queues for accessing network devices and network disk resources; and in web-services stress increases the waiting queues, delivery times and , in general, the throughput of the service. Resources can be modeled individually for each VM in a no-stress context, but when resources are overloaded by

constrained resources or VMs interacting and competing for them, behaviors change and often predict consequences (QoS) is better than predict these changes individually.

5.1 Behaviors on Stress Behavior on load for web servers can be depicted as three phases: a load phase, where available resources are more than the required; a stress phase, where resources are fully occupied and services, requests and data pools compete for accessing to CPU, Memory or IO, degrading their behavior towards the client with high Response Times (RT), low QoS, as throughput can't grow with the load; and a crash phase, where load overwhelms resources and capacity of management degrades and system is unable to provide service. Figure 8 shows the diagram of load versus web-service throughput and QoS.

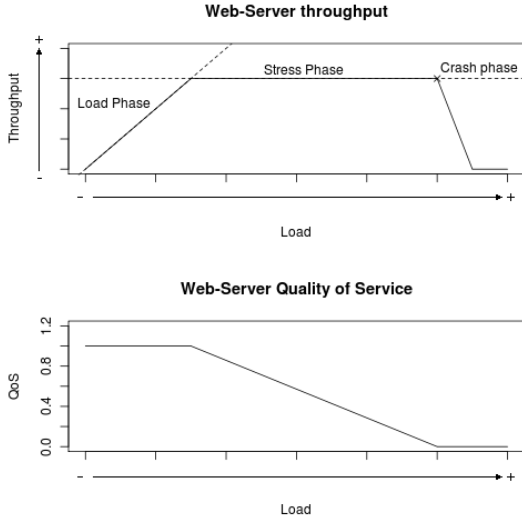


Figure 8: Behavior of Web Services versus Load and Capacity

The resources modeling shown in the previous section correspond to the load phase, where each VM behaves natural without competition. The next step of this work is to learn automatically about behaviors and QoS on stress situation, being able to plan without an excess of resource reservation. Figure 9 shows the behavior of a VM in a PM with increasing CPU/MEM competition from other jobs and VMs, until the point of an aggregated requirement of CPU greater than the available. For the provider, the ideal resource optimization would be to work in the “5:4 CPU” region when expecting an acceptable RT, and reallocate VMs and jobs when predicting a high RT situation.

A common QoS function in SLAs, referring to response times, is to set a threshold for the maximum RT to be accepted as good, often with a tolerance margin of β RT. For linear decreasing margins, an SLA function could be

$$SLA = |1 - \alpha \frac{RT - RT_0}{RT_0}|_0^1; (\alpha = \frac{1}{\beta - 1})$$

where $RT \leq RT_0$ implies $SLA = 1$, $RT \geq \beta RT_0$ implies $SLA = 0$, and RTs between these range decreases the SLA fulfillment linearly. In this case of study we will use this, but other ones could be used depending on the kind of service and the agreement between providers and customers.

5.2 Learning over Stress Having together all the monitored components of the system we can consider this information as the State of the system, in the same way we considered the information about requests as the Load upon the system:

$$Status = \langle CPU\{PM, VM, rVM\}, MEM\{PM, VM, rVM\}, IO\{In, Out\}, \dots \rangle$$

Our main goal, scheduling our jobs in hosts the most efficient way, can be achieved when we are able to place as most VMs in a PM without degrading their QoS. We would like to set PMs to work in states like “5:4CPU” or more while the RT is low, so we can try to use the status and the load to predict this RT and set consolidation configurations without high RTs. So we can try to learn the function:

$$f_{RT}^{learned}(Status, Load) \rightarrow E[RT]$$

In order to learn this function, different techniques can be applied, not only regression ones but also other techniques mapping states to results, like k-NN learners. Unfortunately testing some prediction methods on our data (like Lin-Reg, M5P, SVMs and k-NN) we did not find enough accuracy for working directly with that results (e.g. k-NN: MRE = 0.38; MAE = 0.0068 sec; $\sigma = 0.0228$; Dataset range [0,2.147] seconds). The predictions obtained differentiate between high RTs and low RTs, but precision at calculating RT is quite inaccurate.

As shown before, RT is used as a factor in a SLA function, and if $RT \geq RT_0$ (RT_0 is agreed by provider and customer) the provider must pay a penalty (also agreed in the SLA). Having the RT function so disperse, a solution is to learn the SLA function instead, a function that will set a range to RT. Applying the described

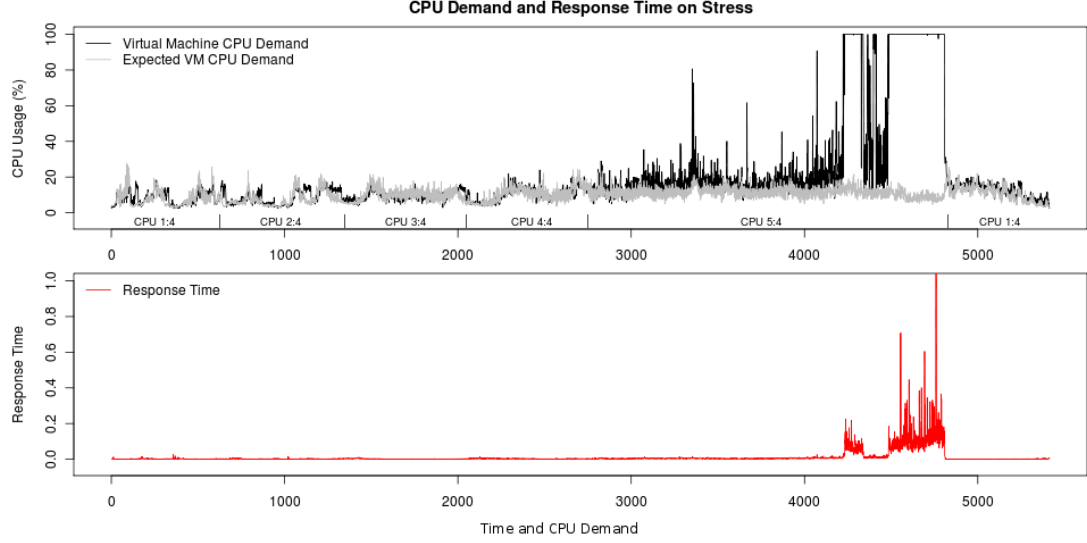


Figure 9: VM behaviors and predictions on Unstressed vs Stressed CPU

SLA over the RT data, with an acceptable RT threshold 0.008 and β 5 for our monitored information, we obtain a new derived attribute “SLA fulfillment”. Now the SLA function ranges the RT, so we try to learn directly the function:

$$f_{SLA}^{learned}(Status, Load) \rightarrow E[SLA]$$

Applying again the M5P technique (among others without so good results), an interesting thing appears in our learned model. As seen in Figure 10 some *Status* are really similar but have different RT. Depending on the load and stress of our host machine (the status), the chances of having instances with good or bad RTs increase and decrease, but a single instance can not be predicted by itself. As seen in the figure, what the model really learns is that in the load phase, the chances of good SLA (low RT) are around 100% and when the host performs memory operations (in this case) SLA averages 80% (this means, 80% of the instances will have $SLA \sim 1$ and 20% $SLA \sim 0$); also it learns that in stress phase (slight heavy load) chances of good SLA are $\in [50, 80]\%$, as near half instances will solve their SLA as 1, and the others as 0; and finally that in stress phase (heavy load) chances of bad SLA are almost 100%, so it predicts $SLA \in [0, 10]\%$ (as some few instances can have good RT).

This brings us to the conclusion that, by having the same status, the chances of having good or bad SLA can be predicted instead of predicting the SLA for a given instance of the status. This could mean that for our monitoring and point of view the RT is stochastic, and more data from the web-services should be required (data that we would not want to require, in order

to grant privacy inside VMs).

6 Significance: Applying Predictions

Having all these predictions, our scheduling becomes a general allocation problem, where the function to be optimized is the profit obtained from the revenue per VM, less the QoS penalties and power costs.

$$Profit = \sum (Revenue(VMs) - Penalty(VMs)) - \sum Power(PMs)$$

Having as input the characteristics of the load, we can predict the amount of CPU, memory and network to be used by each VM. Using this information, a scheduler can try to find the optimal combination of VMs in PMs, and predict the quality of service obtained by the resulting combinations. From here on the scheduler can attempt to reduce VMs given resources in order to make room for more VMs, predicting each time the QoS and trying to not degrade it in excess.

In one of our last works like [3] we developed a mathematical model to maximize this profit taking into account revenues, power costs, different kinds of penalties in SLAs, and estimations of the RT on VMs with CPU variation only, and results for CPU scheduling where promising, as optimal schedules granted average power savings of $\sim 30\%$ respect to classical methods not using learned knowledge. Now, having the models and predictions of the most important resources, different algorithms like complete solvers or approximate algorithms can try to maximize the use of clouds and data-centers with more accurate information about the behaviors of web-services in VMs given loads and physical ma-

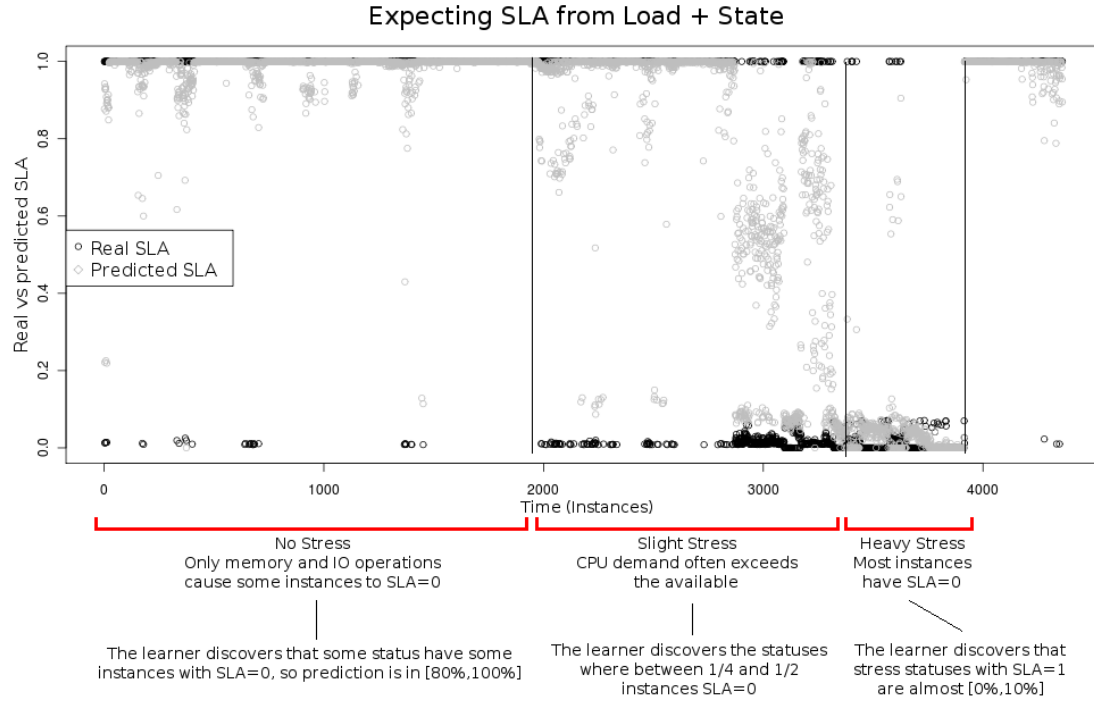


Figure 10: Prediction of SLA using known resource variables

chine contexts

Also with the data-center represented by a mathematical model, when all constraints and models are linear it can be solved using MILP solvers, but as shown here some elements behave non-linear, so other solvers like heuristics approximated algorithms can be used instead. We must take into account that the solving time is important depending on how often we want to re-schedule our data-center, so exhaustive algorithms should be kindly discarded, and alternative algorithms must be also evaluated not only in optimality but also in practical computation time. Part of our next work consists on finding the adequate algorithms to schedule and assign resource quotas, by maximizing the target function using all the valuable information obtained by the learned modules.

7 Conclusions

In this work we presented cloud computing as a new scenario for data mining and machine learning to work with, and the importance to obtain information and knowledge to drive new Internet infrastructures and services appropriately. Learning and modeling the behaviors of cloud elements is a field to be explored and improved using techniques of knowledge discovery and machine learning.

With the purpose of starting improving the cloud management on setting jobs on hosting

machines, we started modeling the resource and cloud jobs (web-service) elements. Jobs submitted to the cloud require resources like CPU, memory and network to work, and cloud managers must ensure the provision of them. Here we modeled these resources behavior and predicted these requirements a priori, so the manager will dispose of this information before planning. Further, we modeled basic relations between low level and high level metrics like resource provision versus quality of service, so the planner can adjust the provision of resources having always job customers satisfied.

In the next steps we will expand the current models to full data-center and full cloud models, to provide decision makers information about data-center interactions and clouds. Also we will study different planning solvers like MILP solvers, heuristics and approximate algorithms, applying the presented models, and used on real data-centers.

References

- [1] RDLab - Department of Software UPC, 2011. <http://rdlab.lsi.upc.edu/>.
- [2] J. Alonso, J. Torres, J. L. Berral, and R. Gavalda. Adaptive on-line software aging prediction based on machine learning. In *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010*. IEEE, 2010.

- [3] J. Berral, R. Gavaldà, and J. Torres. Adaptive Scheduling on Power-Aware Managed Data-Centers using Machine Learning. In *12th IEEE International Conference on Grid Computing (GRID 2011)*, 2011.
- [4] J. Berral, R. Gavaldà, and J. Torres. Li-BCN Workload 2010, 2011. http://www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=1099.
- [5] J. Berral, Í. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres. Towards energy-aware scheduling in data centers using machine learning. In *1st International Conference on Energy-Efficient Computing and Networking (eEnergy'10)*, pages 215–224, 2010.
- [6] J. S. Chase, D. C. Anderson, P. N. Thakar, and A. M. Vahdat. Managing energy and server resources in hosting centers. In *18th ACM Symposium on Operating System Principles (SOSP)*, pages 103–116, 2001.
- [7] G. Dhiman. Dynamic power management using machine learning. In *IEEE/ACM Intl. Conf. on Computer-Aided Design 2006*, 2006.
- [8] Í. Goiri, F. Julià, R. Nou, J. Berral, J. Guitart, and J. Torres. Energy-aware Scheduling in Virtualized Datacenters. In *12th IEEE International Conference on Cluster Computing (Cluster 2010)*, 2010.
- [9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.
- [10] I. Kamitsos, L. Andrew, H. Kim, and M. Chiang. Optimal Sleep Patterns for Serving Delay-Tolerant Jobs. In *1st International Conference on Energy-Efficient Computing and Networking (eEnergy'10)*, 2010.
- [11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [12] Y. Tan, W. Liu, and Q. Qiu. Adaptive power management using reinforcement learning. In *International Conference on Computer-Aided Design (ICCAD '09)*, pages 461–467, New York, NY, USA, 2009. ACM.
- [13] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *In Proc. of ICAC-06*, pages 65–73, 2006.